# Critique of Unconstrained Memoryless Modular Strategies in Continual Learning

Author Name

June 26, 2024

## Abstract

This paper is a critique of unconstrained memoryless modular strategies in continual learning. We argue that these strategies are not well-suited for the problem of continual learning, as they certainly don't solve it but at best, solve specific benchmarks. These approaches do not deal with the core challenges of distributing inputs to modules and merging outputs from modules. More fundamentally, we show that these strategies are inherently unable to tackle these challenges and that they mainly mitigate pollution between tasks. We also discuss the management of hyperparameters in continual learning, which is often unfair and can lead to overfitting on benchmarks. Finally, we propose a naive approach that reaches state-of-the-art performance on two benchmarks, highlighting the limitations of unconstrained memoryless modular strategies.

## 1 Introduction

## 2 Definitions

### 2.1 Continual learning

Continual learning is the problem of learning a sequence of tasks without forgetting the previously learned tasks. Continual learning comes with various flavours such as online learning, lifelong learning, incremental learning, etc. In each of these frameworks, assumptions made are slightly different. For example, in online learning, the data is assumed to be generated by a stationary distribution, whereas in lifelong learning, the data is assumed to be generated by a non-stationary distribution. In incremental learning, the data is assumed to be generated by a sequence of tasks. Further, among incremental learning, one can make the assumption that the information of switching from one task to another is given or not given. One can also assume that total number of tasks at hand is known or unknown.

### 2.2 Unconstrained and memoryless strategies

Litterature identifies various strategies and offer various classification in groups, such as gradient-based solutions, replay-based solutions, regularization-based solutions, etc.

Unconstrained strategies are strategies that do not impose any constraint on the learning process that could allow the processing or implicit remembering of the specificities of the past task. This could be achieved through the a priori knowledge of the meaningful features of all the tasks at hand, of through a parametrizations of the differences between the tasks.

Memoryless strategies are strategies that do not use any memory of the past tasks.

### 2.3 Modularity

Another category of strategies that we did not mentioned in the previous section is the modular strategies. Modular strategies are strategies that decompose the learning process into subproblems. These subproblems might be solved independently, they might consist in processing different features.

In addition to the processing of the subproblem, modular approaches require to solve two challenge. First, after deployement, when given a sample belonging to one of the tasks the model has been trained on comes, it has to flow properly trhough the modular components. Secondly, the solutions of the subproblems $i.e.$ the outputs of each modular component, have then to be combined meaningfully to form the solution of the original problem.

### 2.4 Unconstrained Memoryless Modular Strategies in Continual Learning

Now we have all the tools to define the object of our criticism. Unconstrained Memoryless Modular Strategies in Continual Learning are strategies that do not impose any constraint on the learning process that could allow the processing or implicit remembering of the specificities of the past task, that do not use any memory of the past tasks, and that decompose the learning process into subproblems. Unconstrained property imposes that these subproblems can't task specific pre-defined a priori by human. They have to be built implicitly through the learning process by the model itself.

This includes the following strategies such as Elastic Weight Consolidation (EWC),

## 3 Litterature reviews on state of the art

Litterature reviews on continual learning are mostly descriptive and not critical enough.

## 4 Our criticism

### 4.1 Pollutions

Here we will introduce a formalism for pollution, add a plot

As a very general principle, the problem with these approaches in the context of continual learning is that

*Improving on task 2 mean regressing on task 1, and reciprocally*

Which implies that performing continual learning in such a way requires to perform trade-offs. And the most convenient way one performs trade-offs in deep learning is through hyperparameters.

We often assume that it is fine to attempt somewhat limited approaches because improvements are achieved by building on failed attempts. This is true for incremental improvements, typically made by engineers through the combination and optimization of previous work. However, researchers aim for disruptive improvements by stepping back from the current state of the art to identify what is arbitrary, poorly motivated, or inherently limited in current approaches.

Let's assume we are given $I$ tasks indexed by the set $[\![1, I]\!]$. We define a neural network as the set of its parameters

$$\phi = \{\{w_{l,m,n} \in \mathbb{R} | m, n \in [\![1, N_{l-1}]\!] \times [\![1, N_l]\!]\} \, | l \in [\![1, L+1]\!]\} \tag{1}$$

where $L$ is the number of hidden layers and $N_l$ is the number of neurons on layer $l$.
Let's define $\mathbb{W}^i_{l,n}$ to be the set of protected weights for task $i$ with respect to neuron $n$ of layer $l$, for any $i \in [\![1, I]\!]$, $l \in [\![1, L+1]\!]$ and $n \in [\![1, N_l]\!]$:

$$\mathbb{W}^i_{l,n} = \{w_{l,m,n}, m \in [\![1, N_{l-1}]\!] | w_{l,m,n} \text{ protected}\} \tag{2}$$

which allows to define the pollution of task $i$ by protected weights for task $j$ on neuron $n$ of layer $l$ as

$$p^{i \to j}_{l,n} = \mathbb{E}_{x \sim D_i} \left[ \frac{\sum_{w \in \mathbb{W}^j_{l,n}} w(x)}{\sum_{w \in \mathbb{W}^i_{l,n}} w(x)} \right] \tag{3}$$

Finally, we can define the pollution of task $i$ by protected weights for task $j$ on the whole network as

$$P^{i \to j} = \frac{1}{L+1} \sum_{l=1}^{L+1} \frac{1}{N_l} \sum_{n=1}^{N_l} p_{l,n}^{i \to j} \tag{4}$$

To this regard, as a general concept, we can control :

- how the added neurons react to new task samples
- how the old neurons react to new task samples
- (depending on the approach) how the old neurons react to previous tasks samples

However, we have absolutely no control over the way the added neurons react to previous tasks samples because :

- we have no acces to previous tasks samples
- we are not using any rehearsal strategy
- we do not intend to encode any information about the specificity of previous tasks in the weights

So our very last hopes without memory is that the weights are such that they encode information about the task's specificities. But, we considere unconstrained approaches, so the model has no reason to encourage such behavior.

## 4.2 Hyperparameters (mis)manipulations in continual learning

Continual learning enables several ways to manipulate hyperparameters. In this subsection, we will discuss various approaches to manipulate them. The perspective of most papers is to introduce a hyperparameter at the task-level, generaly in the loss used to train on a task. That is, they write the loss used while training on task 1 and the loss used for training on task 2, where they introduce the hyperparameter. This hyperparameter appears as an attempt to control the trade-off between the importance of the new task and the importance of the old tasks. However, they remain fuzzy about what happen to this hyperparameter on the latter tasks and the way to choose the value for the hyperparameter.

Let's introduce a hyperparameter $\lambda$. One could set this hyperparameter to be the same for all the tasks, which would result in a total of 1 hyperparameter. Alternatively, one could set this hyperparameter to a specific value $\lambda_i$ for each task $i > 1$, which would result in a total of $n - 1$ hyperparameters. Or one could even set a different value for this hyperparameter for each pair of tasks, meaning that one could introduce $\lambda_{i,j}$ for mitigating catastrophic forgetting of task $j < i$ when learning task $i > 1$. This would result in a total of $\frac{(n-1)n}{2}$ hyperparameters $i.e.$ $O(n^2)$ hyperparameters. Thus, it appears that the fuzziness of authors about the precise number of hyperparameters is not a minor issue as it can lead to very different hyperparameters manipulation.

Not only is the number of hyperparameters introduced by the authors not clear, but also the way they are chosen. If they introduce only 1 hyperparameter, that is the same for all tasks, then in the best case, it is chosen by hand, which is a very bad practice. In this case, the authors should at least explain why they chose this value or provide a heuristic, which is often not the case. In the worst case, they choose this hyperparameter through a hyperparameter optimization over all the tasks − see [Table 1] top-left − which is highly problematic due to data leakage. This breaks the continual learning framework as, after deployement, one has to decide the value of the hyperparameter when training on task 2 without accessing data from the latter tasks to help deciding. This is considered cheating in the context of continual learning. However, it seems that several papers are doing so. The same could be performed with in the $O(n)$ and $O(n^2)$ hyperparameters scenarios − see [Table 1] top-middle and top-right − which would be even more problematic, but as it looks even more obviously problematic, it seems that no one had the idea to do so.

In the three cases discussed above, we considered a single HPO performed on the training through all tasks simultaneously, which we regard as unfair. An

alternative approach is to perform an HPO for each task individually, in a greedy fashion. This approach assumes at least one hyperparameter by task, so $O(1)$ hyperparameters is not suited − see [Table 1] bottom-left. This approach enables defining $O(n^2)$ hyperparameters $\lambda_{i,j}$ for mitigating catastrophic forgetting of task $j < i$ when learning task $i > 1$, where one performs an HPO on task $i$ once the model is already trained on tasks $j < i$, before moving to task $i + 1$ − see [Table 1] bottom-right. However, we argue that it represents to many parameters, and makes questionnable the sacalability of this approach. Additionnaly, it is likely to lead to an overfitting on benchmark through over-specificity of tradeoffs. Finally the most reasonable approach seems to perform an HPO on each task individually, in a greedy fashion, but with only $O(n)$ hyperparameters, one for each task − see [Table 1] bottom-middle. This approach is compatible with the continual learning framework, as setting the hyperparameter for training on task $i$ can be done without accessing data from the latter tasks.

Actually, what we refere to as "cheated HPO" where a single HPO is performed through all the tasks at the same time, could be just fine if one was performing the HPO on a benchmark, and then show the performances of the approach on another benchmark with the same hyperparameters. But this is not the case, as authors report performance on the benchmarks they performed hyperparameter optimization on. As we mentioned earlier, the role of these hyperparameters is to perform trade-offs between how much the model remembers of each task. So, *apriori*, we argue that there is no reason to believe that the hyperparameters that are good for a benchmark are good for another benchmark as they encapsulate the relative differences of the tasks within the benchmark. This benchmark-specificity of hyperparameters is the source of what we refere to as "overfitting on benchmark". Additionnaly, one should note that the cheated HPO implicitly exploit the number tasks, so the values of the hyperparameters obtained through this HPO would only be suited for benchmarks containing the same amout of tasks, which offers very narrow deployement opportunities.

A relevant inquiry would be to quantify this overfitting. An approch to do so is to perform HPO on a benchmark and then measure the drop of performance on another benchmark. Our hypothesis is that we should observe very little drop of performance between two benchmarks made of 10 different p-mnist task. However, if we decide of the hyperparameters on a benchmark made of 10 p-MNIST tasks and then test the strategy on a benchmark made of 10 soft $8 \times 8$-p-MNIST task, as defined in [8], we expect a drop in perfomances. This drop should be even more visible if test benchmark is derived from a different dataset, such as CIFAR-100. Note that, with the willingness to be very careful and precise, a proper quantification of overfitting of a strategy should be considered in relation to the similarity between benchmarks. Indeed, it is reasonnable that two very different benchmarks require two different sets to of hyperparameters. Unfortunately, this requires to define and quantify similarity between tasks, and it is a subtle inquiry to build a relevent metric that is versatile enough.

Finally, it appears that manipulation of hyperparameters in the context of continual learning is subtle, and we encourage more care in their management. Additionnaly, hyperparameters keep being thought at the scale of tasks, but instead of task-wise hyperparameters, a with thinking hyperparameters at the scale of the entire benchmark could be an interesting new perspective, more aligned with the continual learning framework. However, this would require to conducts HPOs on one or multiple benchmarks and evaluate the performance on a different benchmark using the same set of hyperparameters, to prevent overfitting on benchmarks.

### 4.3 An example of specious paper

A good example of paper that encapsulate all these issues :

- No code available. Even though the community reimplemented the method, the fact that atuhors do not publicly release the code is still an issue because when reproducing the code, one introduces the hyperparameters but cannot replicate the way have been manipulated by authors.

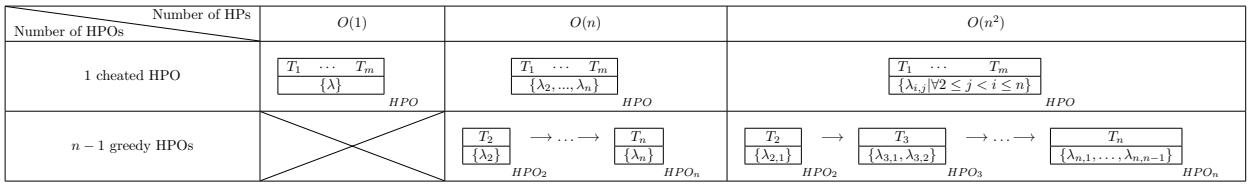



Table 1: HPOs in continual learning

- Authors introduce the method and even explain it in greate details [1] for 2 tasks, but they don't discuss the challenges of scaling it to more tasks. AS a consequence, we don't know what is the loss used for training on latter tasks. They simply write, without more details

  > *"This can be enforced either with two separate penalties, or as one by noting that the sum of two quadratic penalties is itself a quadratic penalty"*

  which is ambiguous and insufficient to be clear about the number of hyperparameters introduced by their method over the entire continual learning process.

- In appendix, authors provide a table containing the value of hyperparameters. The table is quite exhaustive, except when it comes to $\lambda$, which is the hyperparameter they introduced through their method.

- As we demonstrated above, it is not enough to declare the values of the hyperparameters. However, when it comes to explain why or how these values were chosen, authors are fuzzy once again. They write

  > *When random hyperparameter search was used, 50 combinations of parameters were attempted for each number experiment*

  which does not enables to identify which of the 5 approaches to HPO they used, of if they used another approach.

- Despite this paper having 14 authors, it contains several inconsistencies in the name of its tables and figures. In section 4.1, about MNIST experiments, authors refere to Figure 3, which summarizes results of the reinforcement learning experiment. Conversely, Table 2, containing the value of hyperparameters for the reinforcement learning experiment, is entitled "Hyperparameters for each of the MNIST figures".

Le bail avec les élipses là : c'est joli, mais est ce qu'on a des garanties sur la distance des minimum de chaque tâche ? Parce que donc ça représente le minimum du modèle sur données de task 1 uniquement, et sur données de task 2 uniquement. Mais quid du minimum sur données de task 1 et 2 à la fois (parce qu'au fond, c'est ce qu'on essaye d'atteindre, on peut guère faire mieux nan ?) ?

Dessin avec 4 elipses : l'orientation des elipse est quand même très idéale mdrrrr

Encore une fois, il fait une étude théorique très complète pour 2 tâches mais ça ne nous est guère utile en pratique

NON, plus subtile, curvature : De ce qu'on comprend, la Fisher information est simplement utilisée comme une métrique pour dire que le meilleur chemin pour migrer vers $\theta^*_B$ est celui qui nous maintient le plus longtemps dans le minimum pour la tache A

Si on écrit la phrase suivante, on doit être capable d'atteidre SoTA avec juste tradeoffs de modèles entrainés en // : EWC is nothing more than a way to incorporate tradeoffs between tasks in the differentiable optimization process.

Ils ne questionnent pas la légitimité de cette hypothèse : This is with the assumption that there is always an overlapping region for the solution spaces of all tasks for the network to learn them sequentially.

Ils ne contrôlent pas 2 choses : - la distance entre zone acceptable pour tâche A et zone acceptable pour tâche B, donc ils ne contrôlent pas à quel point on va perdre en performance par rapport à ce qu'on pourrait atteindre sur chaque tâche individuellement - ecart entre la valeur du minimum visé par leur méthode de déplacement vers zone acceptable pour B et valeur du minimum atteint pour l'entrainement conjoint sur tâche A et B.

On aimerait certainement entrainer le modèle sur toutes les tâches à la fois, et voir les perfs qu'il obtient, on estime que ça constitue un peu une upper bound de ce qu'on peut atteindre, puisque le continual Learning force une exploration greedy de l'espace des paramètres tandis qu'un entrainement sur toutes les tâches simultanément offre une exploration de l'espace des paramètres plus complet. On veut forcer les paramètres à rester près de $\theta_A^*$, mais les paramètres qui marchent le mieux pour les tâches A et B sont probablement très différents de ceux qui marchent bien pour A seulement.

Avec le dessin des elipses, ils suggèrent très clairement, qu'après s'être entrainés sur la tâche A, on reste à proximité du minimum local sur lequel on a finit l'entrainement. Est-ce que c'est le cas (prcq la loss augmente probablement très fortement quand on passe à la tâche B, donc on est propulser bien loin de notre minimum pour la tâche A, cf notre dessin, et pour que ce ne soit pas le cas, il faudrait fortement diminuer le Learning rate, ce qu'il ne font pas, mais de toute manière ils ne mesurent pas l'augmentation de la loss et n'ont donc aucune chance de controler à quel point ils sont envoyés loin de $\theta_A^*$, et donc, a fortiori, de controler le Learning rate) ? Est-ce qu'on a besoin que ce soit le cas (peut être qu'on ne comprend pas bien comment ça marche, et que l'image n'est là qu'à des fins d'illustration?)

Pourquoi est-ce que c'est un trade-off ? Parce qu'on ne se dirige pas vers un minimum du modèle entrainé sur les données de A et B conjointement, mais vers un minimum du modèle entrainé sur les données de B uniquement, initialié d'une certaine manière, et avec la contrainte qu'on ne veut pas s'éloigner trop des valeurs initiales.

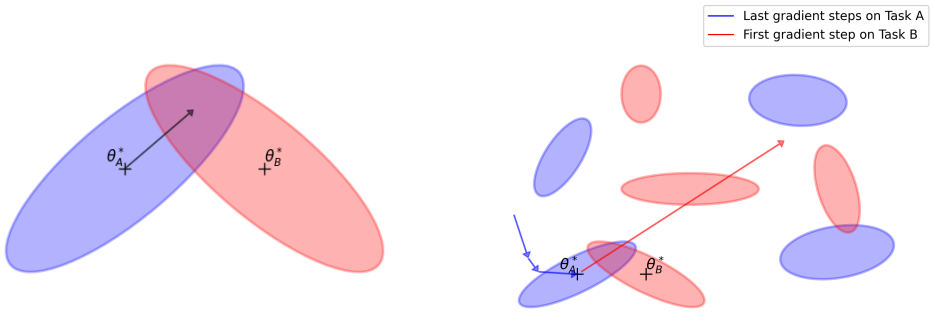
Hum, ya toujours un problème avec les élipses sur lequel on n'a pas encore bien mis le doigt : combien y a t il de loss en jeux ? Et avec quel loss est représenté chaque élément ?

- Elipse A : $L_A$
- Elipse B : $L_B$
- Flèche

EWC is good at specializing modules (subset of weights), but it does not tackle the problem of distributing intputs to the modules and merging outputs from the modules. Which are the core challenges of continual Learning. If distribuing and merging were not problems, we would simply train separated neural networks, one for each task.

Overcomplexification : In AFEC: Active Forgetting of Negative Transfer in Continual Learning, they introduce a new term in the loss, with a new hyperparameter "while the forgetting factor regulates a penalty to selectively merge the main network parameters with the expanded parameters, so as to learn a better overall representation of both the old tasks and the new task.", so they are trying to mitigate the mitigation made by a regularization method such as EWC. Additionnaly, they hope "to learn a better overall representation of both the old tasks and the new task", which has no reason to happen without enforcing such behavior through constraints, which they don't do.

There is no doubt that EWC is capable of solving benchmarks, but with all these inaccuracies and uncertainties, it certainly does not solve continual learning by "overcoming catastrophic forgetting". At best, it mitigates it, and the purpose of the following section is to attempt to quantify how much EWC is able to mitigate catastrophic forgetting.




(a) 2 ellipses as depicted in EWC-like papers (b) A more realistic view of parameters space

Figure 1: Representations of the parameters space

# 5 Methods

## 5.1 EWC

## 5.2 Our new approach (GroHess or naive)

We will also introduce an obviously bad method (or GroHess) that reaches state of the art on 2 benchmarks.

# 6 Discussion

## 6.1 Criticism of the benchmark-oriented approach

We are really good at designing cute benchmarks, like permuted MNIST, which constitute interesting puzzles to solve from the point of view of artificial learning.

but we are not good at solving real-world problems.

## 6.2 Continual learning usecases

We already argued that continual learning is not a well-defined problem. In this section, we will question the very fundamental assumption of continual learning. and it is not clear what are the usecases of continual learning.

Here is a list each and every usecase of continual learning :

- Adapt to new data quickly https://neptune.ai/blog/continual-learning-methods-and-application
  - Bank fraud (want to adapt quickly to new method)
- A model needs to be personalized https://neptune.ai/blog/continual-learning-methods-and-application
  - Let's say you maintain a document classification pipeline, and each of your many users has slightly different data to be processed—for example, documents with different vocabulary and writing styles. With continual learning, you can use each document to automatically retrain models, gradually adjusting it to the data the user uploads to the system.

Remarks :

- For bank fraud, we don't have enough knowledge to formulate objections.

- Model personalization sounds like finetuining, not continual learning... does it make sense to try to train a single model that we want to use on every client ? Isn't it better to do it in the LoRA fashion, and fine-tune a module that we store with the client's data ?

# 7 Conclusion

In the end, this efforts put in writting this paper beautifully illustrate Brandolini's law :

> The amount of energy needed to refute bullshit is an order of magnitude bigger than that needed to produce it

We corrected the work subsequent to two small mistakes, namely, that could have been avoided by a more careful preliminary analysis of the problem. The fact that a method relies on an attempt to balance how much we hurt the learning process on each task naturally introduces trade-offs. These trade-offs are performed through hyperparameters in the loss that set the relative importance of the task, which can easily be mismanipulated or lead to overfitting on the benchmarks at hand.

# References


[1] Abhishek Aich. *Elastic Weight Consolidation (EWC): Nuts and Bolts*. 2021. arXiv: 2105.04093.

[2] Thomas Dalgaty et al. "Mosaic: in-memory computing and routing for small-world spike-based neuromorphic systems". In: *Nature Communications* 15.1 (Jan. 2024). ISSN: 2041-1723. DOI: 10.1038/s41467-023-44365-x. URL: http://dx.doi.org/10.1038/s41467-023-44365-x.

[3] David Eigen, Marc'Aurelio Ranzato, and Ilya Sutskever. *Learning Factored Representations in a Deep Mixture of Experts*. 2014. arXiv: 1312.4314 [cs.LG].

[4] Jason K. Eshraghian et al. *Training Spiking Neural Networks Using Lessons From Deep Learning*. 2023. arXiv: 2109.12894 [cs.NE].

[5] Utku Evci et al. *GradMax: Growing Neural Networks using Gradient Information*. 2022. arXiv: 2201.05125 [cs.LG].

[6] Raia Hadsell et al. "Embracing Change: Continual Learning in Deep Neural Networks". In: *Trends in Cognitive Sciences* (Nov. 2020). Open Access, Creative Commons Attribution – NonCommercial – NoDerivs (CC BY-NC-ND 4.0). DOI: 10.1016/j.tics.2020.09.004. URL: https://doi.org/10.1016/j.tics.2020.09.004.

[7] Ronald Kemker et al. *Measuring Catastrophic Forgetting in Neural Networks*. 2017. arXiv: 1708.02072.

[8] James Kirkpatrick et al. "Overcoming catastrophic forgetting in neural networks". In: *Proceedings of the National Academy of Sciences* 114.13 (Mar. 2017), pp. 3521–3526. ISSN: 1091-6490. DOI: 10.1073/pnas.1611835114. URL: http://dx.doi.org/10.1073/pnas.1611835114.

[9] Dhireesha Kudithipudi et al. "Biological underpinnings for lifelong learning machines". In: *Nature Machine Intelligence* 4.3 (Mar. 2022), pp. 196–210. ISSN: 2522-5839. DOI: 10.1038/s42256-022-00452-0. URL: https://doi.org/10.1038/s42256-022-00452-0.

[10] Dhireesha Kudithipudi et al. "Design principles for lifelong learning AI accelerators". In: *Nature Electronics* 6.11 (Nov. 2023), pp. 807–822. ISSN: 2520-1131. DOI: 10.1038/s41928-023-01054-3. URL: https://doi.org/10.1038/s41928-023-01054-3.

[11] Axel Laborieux et al. "Synaptic metaplasticity in binarized neural networks". In: *Nature Communications* 12.1 (May 2021), p. 2549. ISSN: 2041-1723. DOI: 10.1038/s41467-021-22768-y. URL: https://doi.org/10.1038/s41467-021-22768-y.

[12] Ziming Liu et al. *Growing Brains: Co-emergence of Anatomical and Functional Modularity in Recurrent Neural Networks*. 2023. arXiv: 2310.07711 [q-bio.NC].

[13] German Ignacio Parisi et al. "Continual Lifelong Learning with Neural Networks: A Review". In: *CoRR* abs/1802.07569 (2018). arXiv: 1802.07569. URL: http://arxiv.org/abs/1802.07569.

[14] Javier Lopez Randulfe and Leon Bonde Larsen. *A multi-agent model for growing spiking neural networks*. 2020. arXiv: 2010.15045 [cs.NE].

[15] Liyuan Wang et al. *AFEC: Active Forgetting of Negative Transfer in Continual Learning*. 2021. arXiv: 2110.12187.

[16] Xin Yuan, Pedro Savarese, and Michael Maire. *Accelerated Training via Incrementally Growing Neural Networks using Variance Transfer and Learning Rate Adaptation*. 2023. arXiv: 2306.12700 [cs.LG].